UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/900,123	07/05/2001	Scott Wiltamuth	MSFT-0573/160076.1	5765

41505 7590 01/30/2008
WOODCOCK WASHBURN LLP (MICROSOFT CORPORATION)
CIRA CENTRE, 12TH FLOOR
2929 ARCH STREET
PHILADELPHIA, PA 19104-2891

EXAMINER	
KHATRI, ANIL	
ART UNIT	PAPER NUMBER
2191	
MAIL DATE	DELIVERY MODE
01/30/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	09/900,123	WILTAMUTH ET AL.	
	Examiner	Art Unit	
	Anil Khatri	2191	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on 30 July 2007.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) 1-20,22-41 and 58-76 is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) 1-20,22-41 and 58-76 is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

Claim Objections

Claim 74 is objected to because of the following informalities: claim 74 depends on canceled claim 54. Appropriate correction is required.

Claim Rejections - 35 USC § 103

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

Claims 1-20, 22-41 and 58-76 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Aptus et al* USPN 6,993,759 in view of *Sweeney et al* USPN 6,230,314

Regarding claims 1, 23 and 61

Aptus et al teaches,

utilizing an explicit interface member mechanism that enables at least one software component to implement at least one explicit interface member by explicitly specifying the relationship between at least one software component and the at least one explicit interface member (column 14, table 10, also some compilers run faster with the abbreviated assignment operator. Use `this` Explicitly Tries to make the developer use `this` explicitly when trying to To Access Class access class members. Using the same class member names with Members parameter names often makes what the developer is referring to) ; and storing at least one software component in a form that includes implemented at least one explicit interface member in a computer readable storage medium (column 14, table 10, also some compilers run faster with the abbreviated assignment operator. Use `this` Explicitly Tries to make the developer use this` explicitly when trying to To Access Class access class members. Using the same class member names with Members parameter names often makes what thedeveloper is referring to). Aptus et al does not teach explicitly specifying the relationship between at least one software component. However, Sweeney et al teaches (column 9, lines 57-63, A set of type constraints is computed in step 703. These constraints capture the subtype-relationships between variables and members that must be retained by the specialized class hierarchy, which is discussed below, in order to preserve the program's behavior. Preferably, such type constraints are represented by 3-tuples of the form <S,n,T> where S and T are sets with elements of the form). Therefore, it would have been obvious to a person of ordinary skill in the art at the time of the invention was made to incorporate member relationship in component. The modification would have been obvious because one of ordinary skill in the art would have been motivated to combine teaching into member interface in software component to develop new software faster and apply reuse approach to develop product faster.

Regarding claims 2, 24, 62 and 74

Sweeney et al teaches,

specifying of the relationship includes specifying a qualified name of the at least one software component (column 6, lines 27-42, FIG. 7 illustrates the operation of the class hierarchy specialization technique of the present invention. In step 701, basic information about the program and the class hierarchy is collected. A more detailed description of step 701 is described below with respect to FIGS. 8-12. In step 703, a set of type constraints is constructed that capture the required subtype-relationships between the types of objects, and the visibility relations between members and objects that must be retained in order to preserve program behavior. A more detailed description of step 703 is described below with respect to FIG. 13. In step 705, the type constraints computed in step 703 are used to construct a new class hierarchy, and the variable declarations in the program are changed to take into account this new hierarchy. Step 705 preferably consists of three sub-steps).

Regarding claims 3, 25 and 63

Aptus et al teaches,

specifying of the qualified name includes specifying at least one interface name and at least one interface member name (column 14, table 10).

Regarding claims 4, 26 and 64

Aptus et al teaches,

explicit interface member mechanism enables an explicit interface member implementation to be excluded from the public interface of at least one software component (column 22, lines 50-63, There is also a fourth view mingled with the static view called the architectural view. This view is modeled using package, component and deployment diagrams. Package diagrams show packages of classes and the dependencies among them. Component diagrams 1800, depicted in FIG. 18, are graphical representations of a system or its component parts. Component diagrams 1800 show the dependencies among software components, including source code components, binary code components and executable components. As depicted in FIG. 19, deployment diagrams 1900 are used to show the distribution strategy for a distributed object system. Deployment diagrams 1900 show the configuration of run-time processing elements and the software components, processes and objects that live on them).

Regarding claims 5, 27, 65 and 71

Aptus et al teaches,

the at least one software component is at least one of a class or struct instance, as defined by the object-oriented programming language (columns 18-19, table 17).

Regarding claims 6, 28, 66 and 75

the explicit interface member mechanism enables at least one software component to implement an internal interface not accessible to a consumer of at least one software component (table 17).

Regarding claims 7, 29 and 67

Aptus et al teaches,

explicit interface member mechanism enables disambiguation of a plurality of interface members having the same signature (columns 14-15, see table 11)

Regarding claims 8, 30 and 68

Aptus et al teaches,

explicit member mechanism enables disambiguation of a plurality of interface members having the same signature and return type (columns 14-15, table 11).

Regarding claims 9, 31 and 69

Aptus et al teaches,

in addition to allowing the implementation of public interface members, explicit interface member mechanism enables the implementation of private interface members (see tables 12, 16 and 17).

Regarding claims 10, 32 and 70

Aptus et al teaches,

explicit interface member mechanism enables the implementation of a plurality of non-conflicting specific versions of a generic interface (column 2, lines 54-63, The software development tool also includes a version control system that permits multiple programmers to work simultaneously on a single software project by maintaining a central repository containing a master copy of a software project and by managing versions of the software project that the programmers develop during the development process. The software development tool enables programmers to interact with the version control system by manipulating a diagram that corresponds to the software project, thus facilitating the use of the version control system).

Regarding claims 11 and 33

Aptus et al teaches,

the computer code is programmed according to an object-oriented programming language, and object-oriented programming language is one of C#, Fortran, Pascal, Visual Basic, C, C++ and Java (column 21, lines 1-37, FIG. 9 depicts a flow diagram of the steps performed by the software development tool to develop a project in accordance with methods consistent with the present invention. As previously stated, the project comprises a plurality of files. The developer either uses the software development tool to open a file that contains existing source code, or to create a file in which the source code will be developed. If the software development tool is used to open the file, determined in step 900, the software development tool initially determines the programming language in which the code is written (step 902). The language is identified by the extension of the file, e.g., ".java" identifies source code written in the Java.TM. language, while ".cpp" identifies source code written in C++. The software development tool then obtains a template for the current programming language, i.e., a collection of generalized definitions for the particular language that can be used to build the data structure (step 904). For example, the templates used to define a new Java.TM. class contains a default name, e.g., "Class1," and the default code, "public class Class1 [ ]." Such templates are well known in the art. For example, the "Microsoft Foundation Class Library" and the "Microsoft Word Template For Business Use Case Modeling" are examples of standard template libraries from which programmers can choose individual template classes. The software development tool uses the template to parse the source code (step 906), and create the data structure (step 908). After creating the data structure or if there is no existing code, the software development tool awaits an event, i.e., a modification or addition to the source code by the developer (step 910). If an event is received and the event is to close the file (step

912), the file is saved (step 914) and closed (step 916). Otherwise, the software development tool performs the event (step 918), i.e., the tool makes the modification. The software development tool then updates the TMM or model (step 920), as discussed in detail below, and updates both the graphical and the textual views (step 922).

Regarding claims 12, 34 and 72

Aptus et al teaches,

an implementation of an explicit interface ember is a method, property, event, or indexer declaration that references a fully qualified interface member name (column 21, lines 1-37, FIG. 9 depicts a flow diagram of the steps performed by the software development tool to develop a project in accordance with methods consistent with the present invention. As previously stated, the project comprises a plurality of files. The developer either uses the software development tool to open a file that contains existing source code, or to create a file in which the source code will be developed. If the software development tool is used to open the file, determined in step 900, the software development tool initially determines the programming language in which the code is written (step 902). The language is identified by the extension of the file, e.g., ".java" identifies source code written in the Java.TM. language, while ".cpp" identifies source code written in C++. The software development tool then obtains a template for the current programming language, i.e., a collection of generalized definitions for the particular language that can be used to build the data structure (step 904). For example, the templates used to define a new Java.TM. class contains a default name, e.g., "Class1," and the default code, "public class Class1 [ ]." Such templates are well known in the art. For example, the "Microsoft Foundation Class Library" and the "Microsoft Word Template For Business Use Case Modeling" are examples of standard template libraries from which programmers can choose individual template classes. The software development tool uses the template to parse the source code (step 906), and create the data structure (step 908). After creating the data structure or if there is no existing code, the software development tool awaits an event, i.e., a modification or addition to the source code by the developer (step 910). If an event is received and the event is to close the file (step 912), the file is saved (step 914) and closed (step 916). Otherwise, the software development tool performs the event (step 918), i.e., the tool makes the modification. The software development tool then updates the TMM or model (step 920), as discussed in detail below, and updates both the graphical and the textual views (step 922).

Regarding claims 13, 35 and 73

Sweeney et al teaches,

At least one software component names an interface in the base class list of the at least one software component that contains a member whose fully qualified name, type, and parameter types exactly match those of the implementation of the explicit interface member (column 16, lines 8-19, In Rule 1 of FIG. 20, the "merging" of two classes X and Y (where X is a base class of Y) involves the following steps: 1 A new class Z is created that (virtually, nonvirtually) inherits from all (virtual, nonvirtual) base classes of X and Y, and that contains all members of X and Y,

2. Each virtual class Z' that inherits from X or Y is made to inherit from Z instead. This inheritance relation is virtual if the inheritance relation between X and Y or the inheritance relation between Y and Z' is virtual; otherwise it is nonvirtual.

Regarding claims 14 and 36

Sweeney et al teaches

explicit interface member mechanism includes an interface mapping mechanism that locates implementations of interface members in at least one software component (column 21, claim 7, mapping said class hierarchy H of said program P and said set of type constraints to nodes of a first sub-object graph; and analyzing nodes and edges of said first sub-object graph to generate a second sub-object graph, wherein said new class hierarchy H' of said program P is based upon said second sub-object graph).

Regarding claims 15 and 37

Sweeney et al teaches

interface mapping mechanism locates an implementation for each member of each interface specified in the base class list of the at least one software component (column 21, claim 7, mapping said class hierarchy H of said program P and said set of type constraints to nodes of a first sub-object graph; and analyzing nodes and edges of said first sub-object graph to generate a second sub-object graph, wherein said new class hierarchy H' of said program P is based upon said second sub-object graph).

Regarding claims 16, 38 and 76

Sweeney et al teaches

at least one software component inherits all interface implementations provided by its base classes (column 4,lines 1-5, The inheritance relations among the classes; in particular, for every class, the set of base classes it inherits from is specified. The inheritance may be further qualified as virtual or non-virtual inheritance.

Regarding claims 17, 39, 58

Aptus et al teaches,

wherein it is not possible to override an explicit interface member implementation, but where an explicit interface member implementation calls another virtual method, derived classes are capable of overriding the implementation (see table 8)

Regarding claims 18, 40 and 59

Sweeney et al teaches

a software component of at least one software component that inherits an interface implementation is permitted to re- implement the interface by including the interface in the base class list of the software component (column 4, lines 1-5, The inheritance relations among the classes; in particular, for every class, the set of base classes it inherits from is specified. The inheritance may be further qualified as virtual or non-virtual inheritance.

Regarding claims 19, 41 and 60

Sweeney et al teaches

explicit interface member mechanism prevents conflict among specific implementations of a generic interface (column 4, lines 35-53, There are some differences between virtual and non-virtual inheritance, which is illustrated by the structure of a D object, as shown in FIG. 4. Consider class D. Since D inherits from B and C, every D object contains a B object as a subobject and a C object as a subobject. Of course, the B subobject contained within a D object will itself, in turn, contain an A subobject and an S subobject. Similarly, the C subobject contained in a D object will itself, in turn, contains an A subobject and an S subobject. However, since B and C inherit virtually from S. both the B subobject contained within a D object and C subobject contained within the same D object will share a single, unique S subobject. In contrast, since B and C inherit non-virtually from A, each of the B and C subobjects contained within a D object contain their own A subobject. FIG. 4 depicts the structure of an object of type D. Observe that, due to a combination of virtual and non-virtual inheritance, and object of type D contains a single subobject of type S, but two distinct subobjects of type A, each of which has a distinct data member x).

Regarding claims 20 and 22

Rejection of claim 1 is incorporated and further claims 20 and 22 cites similar limitation as in claim 1 therefore, claims 20 and 22 are rejected under same rationale.

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Anil Khatri whose telephone number is 571-272-3725. The examiner can normally be reached on M-F 8:30-5:00 PM.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Wei Zhen can be reached on 571-272-3708. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

[illegible]

ANIL KHATRI
PRIMARY EXAMINER